UNIVERSITY OF GOTHENBURG

Editor for Virtual Cars

Configurator for Simulating Electronic Unit Data

Master of Science Thesis in the Programme Computer Science

SAMEH ALNASHI

University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, May 2009

An Editor for Editing Virtual Cars
Configurator for Simulating Electronic Unit Data

SAMEH ALNASHI

Examiner: BJÖRN VON SYDOW

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden May 2009

ABSTRACT

As testing of car electronics now-a-days requires a vast amount of time and can be very costly, car manufacturers have developed a common technology – the virtual platform. Virtual platforms are fast, executable simulations of the hardware and the environment it evolves in. They can represent an entire system, a subsystem or a set of relevant software development functionalities. So the advantage with the virtual platform is to allow a full software load to be run with accuracy enough to produce the true timing and the true network traffic that the real electronics will experience without having to compromise the real electronics. The virtual platform is conceived by logging communication with a real vehicle. Hence, from the vehicle log a virtual platform can be created using a special converting program.

In this thesis a configurator for editing a virtual platform is considered. This means that a program has been written to easily edit, manipulate and change parameters inside a virtual platform. The configurator gives the ability to change the electronic unit types, the communication between the vehicle and the diagnostic application as well as different requests and responses that entail among other things diagnostic trouble codes, parameters and activation codes. These changes are applied with the help of model data, that shows how the virtual platform is built and service data that describes how the communication between a diagnostic system and a physical car is built up.

Preface

This report is my master's thesis, which is the final examination from the Computer Science program at Gothenburg University.

The thesis was conducted at Sörman Information AB in Gothenburg, Sweden.

I would like to express my gratitude to number of people:

Björn von Sydow, my academic supervisor, for taking time to read my thesis and giving me ideas. He has been of great help during the whole project.

My industrial supervisor Jörgen Andersson at Sörman Information AB for making this thesis possible, and for contributing greatly, always taking times to answer my questions. I would especially like to thank my programming supervisor Per Gradeen for the amount of time taken to help me with my questions and also being there for moral support.

Table of Contents

1 Introduction

This chapter presents the background of the thesis and briefly explains the field of vehicle diagnosis. The chapter also describes the problem that is studied, to whom it is targeted and the reason why it is of interest. Furthermore, a brief description of the programming language as well as the programming environment used during the development of this editor is acknowledged. Finally, an outline of the thesis is featured.

1.1 Background

As technology develops, it is possible to create large systems with increasing capabilities that can satisfy the needs of mankind better every day. However, this also means that the complexity of technical systems increases, as does our dependence upon their correct operation. Failure of technical systems around us can in many cases lead to several different damages. This leads to the need of means to prevent essential systems from failing. This can be done using a system, which can detect the presence of a fault before it leads to performance degradation or a system failure. Such a system is called a *diagnostic system.* A diagnostic system can also be able to isolate the faulty component in a larger system. This can help the personnel performing quicker maintenance and thus the return to normal operation of the system.

Early means to diagnose a system was manual inspection. The operator would use his/her senses, such as smelling, looking for anomalous behavior, listening for strange noises and trying to detect abnormal activities. With the introduction of computers and electronics, diagnosis can be performed by checking that certain sensor values are within normal operating range.

In this thesis, *virtual-based diagnosis* is considered. In virtual-based diagnosis, a virtual model of the system is used and in our case it is a virtual car. For example, the virtual model can be fed with the same input as a real car. Doing so will aid technicians to test several different scenarios before applying the real changes to a physical car. Virtual-based diagnosis has a number of advantages compared to traditional diagnosis. Errors can be revoked, will not damage car electronics, reacts faster because communication is not required with the physical car and gives better possibilities for fault isolations.

This master's thesis has been performed at Sörman Information AB and the Computer Science department at Gothenburg University. Sörman is described on their corporate web page as *"…market-leading provider of solutions in the area of after-sales information."* [1].

Sörman develops and markets a software product for diagnosis called UpTime. In UpTime, many different types of diagnostic-related tasks can be performed. Typically, UpTime is used for offline diagnosis. That is, UpTime is normally not connected to the diagnosed system during operation. Instead, it is used when a fault in the systems is already detected.

1.2 Goals and Objectives

The scope of this thesis is to implement an editor, which allows a user to edit a virtual car without having the need for neither manual computations or to look up how the different parts are interpreted. The editing allows the user to simulate different operations with the help of the virtual car. As the virtual car is an XML interpretation of the electronics in a physical car, it is possible to modify parameters inside the virtual car to allow multiple testing scenarios which otherwise would be quite expensive on a real physical car if the electronics would be damaged. The advantage of this editor will allow the user to quickly change or modify parameters, because the editor will list all necessary information. Testing will therefore become much more convenient.

The goal of this thesis was to meet the requirements set up by my industrial supervisor (see Appendix A) and to target the users that will work with UpTime creating virtual cars for simulating different test scenarios when not working with a physical car.

It should also be mentioned that this editor is not a general XML editor, because of the layout that the virtual car is created in. It is created using ODX specified format and therefore the editor is applied only to ODX specified virtual cars.

1.3 Development Tools

In this subchapter, I will briefly write about the different development tools used when developing the virtual car editor. I will mention the programming language used, as well as the development environment.

1.3.1 C-Sharp (C#)

It was decided at the beginning of this project that the editor was to be written in the C# language developed by Microsoft Co. C# is an object-oriented programming language that is part of the .Net-platform. The language is based on the popular languages C and C++ but also have a lot in common with the programming language Java that was developed by Sun Microsystems Inc. The language is intended to be quite simple, modern and general-purpose driven. The C# programming language was chosen because of its simplicity of making an executable file and for easier distribution through Windows computers which leads to taking advantage of distributed environments [2]. One other major reason for choosing C# is, the diagnostic program used to create read-out car logs as well as the converting program that converts read-out car logs to ODX files was developed and implemented in the .Net framework also using C#.

1.3.2 Microsoft .Net Framework

The Microsoft .Net framework is a software framework for handling all aspects of software development. It includes libraries of pre-coded solutions for common

programming problems and a virtual machine[1] that manages the execution of programs written specifically for the framework. The pre-coded solutions that form the Application Programming Interface (API) or else called the Base Class Library (BCL) covers a large range of programming needs in several areas, such as user interface, data access, database connectivity, web application development among other things that are needed for developing an application. It also provides like the Java Virtual Machine a so-called Common Language Runtime (CLR) that deals with functions for exception handling, garbage collection, security and interoperability. When developers develop programs using a .Net language such as C#, a compiler at compile time coverts the code written into Common Intermediate Language (CIL) code even known as byte code. At run-time a CLR just-in-time compiler converts the CIL code into code native to the operating system [3].

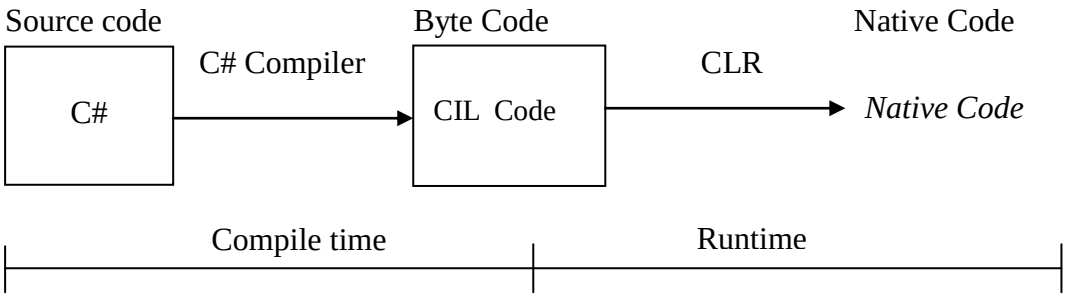


Figure 1.0 – Showing how C# code compiles into native code.

1.3.3 Microsoft Visual Studio

To be able to write C# code in the most convenient way, Microsoft Visual Studio were chosen as an integrated development environment (IDE). Microsoft Visual Studio is used to develop programs such as, console, graphical user interface applications along with other services both in native code and in managed code for all platforms supported by products from Microsoft Co. It includes a code editor for writing code and a debugger that works both as a source-level debugger and a machine-level debugger. Other functionality it possesses is the ability of managing other languages developed by Microsoft Co. and languages that are used through network services such as XML, HTML/XHTML, JavaScript, and CSS. It also has the ability to evaluate regular expression and also supports code refactoring [4].

1.4 Thesis Outline

The thesis is structured as follows: In Chapter 2, the theoretic and background foundation of car technology is laid. A small part describing the intention of the virtual

[1] A virtual machine (VM) is a software implementation of a machine usually a computer that executes programs like a real machine.

car can also be found in chapter 2. Chapter 3 contains a series of information about the virtual car, how it is created and from where. Also, information about what data the user uses to change or modify the content of the virtual car is explained. In Chapter 4, the editor is explored along with the implementation and design of it. The chapter describes how the different design aspects work and what they are suppose to perform. Also described in chapter 4 is the methods used to develop the virtual car editor. Finally, Chapter 5 contains discussion of the result and ideas for improvement for the future.

2 Car Technology

In this chapter the field of car technology and its background is presented. The chapter will explore several technologies used in the car industry, their meaning and the use of them. We will also very briefly look into the different in-car network protocols that are standardized and soon to be by the International Standard Organization. Finally, a new standard called Open Diagnostic Data Exchange, used in the car industry is presented.

2.1 Background

In time, as cars got complex they needed more and more electronic units to serve the vehicle and the driver with several needs. For instance, an electronic unit for the engine now-a-days is a must in every car, due to the fuel emissions and regulations. The brakes are also another example of a vehicle part that needs an electronic unit to control the braking in case of an accident. So the units are used to control several kinds of car parts like the body, temperature adjustment, dashboard, and navigation system, various kinds of sensors, motor and chassis. This gave the car manufacturers ideas to develop several in-car networks and diagnostic protocols to support communication with and between the electronic units. Those ideas paved the way for several industry standards that deals with car communication. Standards such as ISO 14229-1 “Road vehicles – Unified diagnostic services” that gives specification and requirements about the communication and several in-car network protocols for communication between electronic units as well as from the diagnostic system to the units and vice versa (see section 2.6.1). The idea is to make it easy to develop vehicular systems, user interfaces and management for all application domains using these standards. The standards also addresses several important issues like the transfer of information through networks, scalability to different vehicle and platform variants, some basic system functions such as Diagnostic Trouble Codes (see section 2.4) read out, Vehicle Identification Number read out among other things.

2.2 Diagnostic

In early vehicles, diagnostic was just a simple concept. The mechanics or technicians only handled the common mechanical issues when tracing a fault and performing diagnostics on the vehicle. In today’s vehicles the fault tracing is more complicated due to the amount of electronic units used, which operates with advanced software. Almost all functionality of the vehicle is now-a-days controlled by these advanced software ranging from adjusting the AC system to the more advanced engine control systems.

As car manufacturers add a wide range of systems and components that performs various operations while the vehicle is in use, a failure of an individual system or component may occur. Because faults may arise, the vehicle is equipped with an onboard diagnostic computer that communicates with the various systems and components included inside the vehicle. The onboard computer will monitor the operation of the systems and components by logging diagnostic data generated from the

different electronic units, during use of the vehicle. Due to the restriction of the onboard computer, it may not be able to analyze the data to identify the failure that has arisen. Therefore, several diagnostic and support tools have been developed to allow the owner to access and retrieve the diagnostic data logged by the onboard diagnostic computer. Once the data is retrieved, it may be analyzed to determine the failure that occurred.

In the following subsections the diagnostic concept will be more described and cover some subjects that are standardized in the car industry.

2.2.1 On-Board Diagnostic

Before OBD was introduced to the manufacturers, vehicle diagnostics was not always performed in a standardized framework. Manufacturers followed their own way of diagnosis with a customized set of signals. Then in 1998 the Society of Automotive Engineers (SAE) initiated standardization of test signals, which led to more mature standard such as OBD [5].

The faults that may arise when a vehicle is on the road will light a MIL – Malfunction Indicator Lamp – or a Check Engine lamp, which indicates to the driver that a Diagnostic Trouble Code has been set in the memory of the Electronic Control Unit where the fault was caused. The task of OBD is to provide the owner of the vehicle with access to the vehicle's state of health information and the stored diagnostic trouble codes [7]. The OBD implementation provides fast digital communication ports to the real-time data in addition to the standardized series of diagnostic trouble codes, which allows the owner to quickly identify and remedy malfunctions within the vehicle according to different documents from the International Standard Organization [6].

The first version of OBD which was called OBD-I was characterized by simply illuminating the malfunction indicator lamp if a problem was detected – but would not provide any information regarding the nature of the problem. The second implementation of OBD called OBD-II is what most car manufacturers' use today. In addition to the MIL and the check engine lamp - the OBD-II standard defines the type of diagnostic connector and its' pin-out, the electrical signaling protocol and the message format for the protocols. It also manages a candidate list of vehicle parameters to monitor along with the encoding of each DTC [8].

Apart from this, the OBD-II also provides an extensible list of diagnostic trouble codes allowing any diagnostic devices to query the on-board computer in any vehicle. The OBD-II also allows access to numerous data from the electronic control units and offers a valuable source of information when troubleshooting problems inside a vehicle. Finally, the OBD-II protocol can be thought of as a computer based system that monitors virtually every component that can affect the performance of the vehicle to ensure that the vehicle will run as smoothly as possible without any faults, and to assist any vehicle technicians in diagnosing and fixing problems with computerized controls [8].

2.2.2 Diagnostic Programs

In modern vehicles testing and diagnosis of electronic control devices and the interactions between them is more important than ever before. This is needed at every stage of a vehicle's life cycle, from the initial prototype through mass production and on to after-sales service. To this end, specialized diagnostic programs are developed, which must have access to the relevant information about the specific control device used, and thus used by vehicle manufacturers' development, production and service departments.

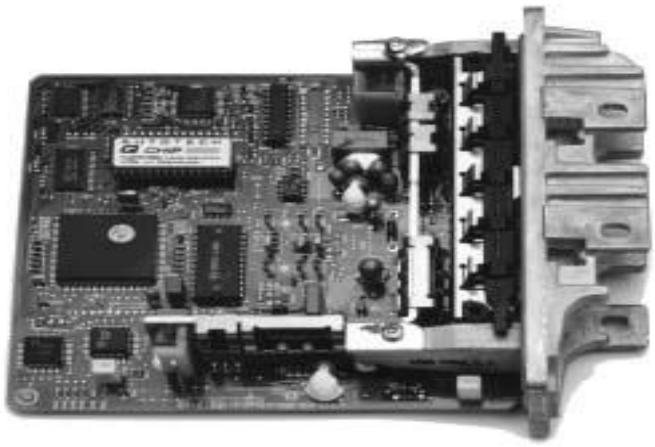
So a diagnostic program is an electronic service and parts manual used for repairing and/or servicing a vehicle, by providing service information, parts information, diagnostic fault tracing and software downloads to the different ECUs integrated into one single application [9].

2.3 ECU – Electronic Control Unit

Electronic control unit, also called a control unit, or a control module, is an embedded system that controls one, or more, of the electrical systems or subsystems in a vehicle. The control unit is the circuitry that controls the flow of information through the processor and coordinates the activities of the other units within it. In a way, it is the "brain within the brain", as it controls the activities inside the processor, which in turn controls the rest of the unit. In other words, the control unit can be thought of as the brain of the CPU itself. It controls, based on the instructions it decodes, how other parts of the unit and in turn, rest of the computer systems should work in order so that the instructions gets executed in a correct manner [9][10].

There are two types of control units. The first type is called hardwired control unit. Hardwired control units are constructed using digital circuits and once formed cannot be changed. The other type of control unit is micro programmed control units. Micro programmed control units itself decodes and executes instructions by means of executing micro programs. This gives the advantage that they can be reloaded by software or upgraded to a newer version of software. This is done in combination with OBD-II (see section 2.2.1) where cars use ECUs that are capable of having their programming changed through an OBD port [11]. Electronic units have become widely used in vehicles, and especially in automobiles. There are various electronic units used in an automobile. For example, an ECU used to control the fuel injection to the engine, an ECU for controlling the transmission and an ECU for handling the anti-lock brake control system among other things. Each ECU works by sending a command signal to the corresponding device and controls the device in a correct manner.
A self-diagnosing function is also provided for each ECU and when an abnormality is detected while controlling the corresponding device, diagnostic data indicating such abnormality is stored in the ECU. Lastly, the ECU can be seen as a device which consists of CPUs and assorted signal input and output dedicated to controlling a component within a vehicle. They range in complexity from an engine control unit which handles power-train system efficiency, to an anti-lock braking control unit that

monitors vehicle speed and brake fluid among other things, to simple body modules that controls doors and windows [12].



Picture 2.0 - A control unit representing an engine control unit.

2.4 Diagnostic Trouble Codes

Computerized control systems can up to a certain point self-diagnose to detect auto problems that could affect the vehicle's emissions and engine performance among other things. So when the control system detects a problem, the control unit will store a trouble code in its memory. A diagnostic trouble code is a special code that is set in the memory of an electronic unit when a fault occurs in any monitored system in the vehicle. The code number corresponds to the type of fault, and can be used to diagnose the problem, and each code number is unique [12][13]. As described previously (see section 2.2.1), when a fault is detected, the electronic unit will store a diagnostic trouble code in its memory and illuminate the MIL or the Check Engine lamp. On some vehicles, the electronic control unit can be put into a special diagnostic mode by grounding certain terminals on an OBD diagnostic connector. This will cause the MIL or the check engine lamp to display the fault code. On many vehicles, though, a diagnostic interface (see section 2.5) must be plugged into the vehicle in order to access and read the fault codes. So in other words the ECU should be able to identify all faults for all components on each of the external circuits connected to the ECU and also all faults internal to the ECU itself [14].

Thus, the basic aim of diagnostics is to detect and report faults in an ECU and its peripherals. Faults can be classified into two categories [14]:

- Intermittent faults that occur only under certain conditions. Since the symptoms of an intermittent fault may not be present during inspection by the car owner, a mechanism is required to store fault related information as soon as the fault is detected, as well as a mechanism to retrieve the particular fault data from the ECU at a later occasion. With help of this stored data, the fault condition can be recreated in order to determine the fault.
- Permanent faults that remain after they first occurred. Tracing permanent faults basically needs a mechanism to read the ECUs' current status (see section 2.5).

An example of a diagnostic trouble code from the site www.obd-codes.com shows how a fault code is structured.

P0100 Mass or Volume Air Flow Circuit Malfunction

Text 2.0 – A representation of a fault code.

This fault indicates that there is a problem with the Mass Air Flow sensor or circuit, which might have been caused by a disconnection in wires or a sensor fault.
The P at the beginning of the trouble code stands for Powertrain and the serial number is unique for each code.

2.5 Vehicle Communication Interface

Vehicle Communication Interface is a device that allows a vehicle technician/mechanic/controller, via a vehicle network, to communicate between a vehicle and a diagnostic program based on the ISO 22900. It is communication apparatus that contains a processor and a field programmable gate array and is used for receiving data transmission from a diagnostic program to be sent to a vehicle, and also to receive and send vehicle communication to the diagnostic program, which then will be logged. The field programmable gate array provides a selectable multiple protocol interfaces that is coupled between the plurality of motor vehicle control units and the processor. The selectable multiple protocol interface converts processor messages into motor vehicle control unit readable formats and converts received control unit information into a processor readable format [14][15].

The information stored in the ECUs in the vehicle must be accessed by remote electronic devices such as development, diagnostic, and software tools during testing and programming phases. The communication interfaces are used to monitor and modify vehicle process variables and other vehicle data during testing and maintenance activities. The vehicle process variables and data indicate if the on-board electronic systems of the vehicle are functioning correctly and also control certain vehicle functions. One of the main uses of a VCI, except for reading out diagnostic trouble codes and among other things, is the possibility of downloading software to the ECUs in the vehicle. The communication interface also supports different kind of in-car network communication making it possible to access the different subnets as well as the regular nets. This makes it able to establish communication with the appropriate vehicle protocol automatically. As described in both OBD-II and the Electronic Control Unit sections (see section 2.2.1 and 2.3), in a typical motor vehicle when a fault occurs, that is monitored by a control unit, the fault is logged within the memory. In attempting to trouble-shoot an indicated fault, a service technician typically connect a vehicle communication interface tool to a diagnostic connector provided on the motor vehicle, for reading out a fault on a particular ECU [15].

2.6 Communication between vehicle and VCI

Using the vehicle communication interface a communication channel will be established between the vehicle and the diagnostic program. As mentioned in earlier sections, several in-car network protocols that are needed for communication might exist. The communication network is utilized to transfer information from an ECU to an off-board tester and vice-versa. The information transferred consists of ECU fault information, parameter reporting, control routines, calibration values and many other types of data used either in performing ECU diagnostics, re-programming or any other data transfer not associated with normal node to node communications [15]. So the communication interface handles transmission and reception of frames from the vehicle buses and it also packs and unpacks signals contained in the frames.

2.6.1 Important Diagnostic Protocols

This section will briefly introduce the different important in-car technologies and protocols used today by several vehicle manufacturers.

- CAN - Controller-Area Network[2] – The CAN protocol is an ISO standard (ISO-11898) for serial data communication. A network protocol and bus standard that allow microcontrollers and devices to communicate with each other without a host computer. It is a broadcast, differential serial bus standard for connecting electronic units in vehicles. It also includes a physical layer and a data-link layer which defines a few different message types, arbitration rules for bus access and methods for fault detection as well as fault confinement. The data-link layer deals with message filtering to decrease the number of unwanted messages entering the buffers and as well as status handling. It also takes care of error detection, transfer rate, and signaling. The physical layer which is the closest layer to the bus deals with the exchange of bits and bytes.

- LIN - Local Interconnect Network – a networking bus-system protocol used within automotive network architectures. The LIN-bus is a small and slow network that is used as a cheap sub-network of a CAN bus to integrate intelligent sensor devices or actuators in vehicles. The network uses serial broadcast from a master to many slaves. There exist no collision detection and therefore all messages are broadcast from the master with at most one slave replying for a given message identifier. The master is typically a commanding microcontroller, whereas the slaves might be less powerful and cheaper microcontrollers.

- MOST – Media Orientation System Transport – is a serial communication system for transmitting audio, video control data via fiber-optic cables

[2] Visit http://www.kvaser.com/can/ for more insight on CAN protocol.

developed by MOST Cooperation[3]. It is intended for interconnecting multimedia components in automobiles and other vehicles. MOST is based on synchronous data communication and defines all seven layers of the OSI reference model. The network employs a ring topology but star configurations and double rings for critical applications are also possible. Due to its' plug-and-play feature it is easy to add and remove a MOST device. Because a MOST network employs ring topology, one MOST device is appointed as Timing Master which continuously feeds data frame into the ring or acts as a gate for data. The packet header that is sent around synchronizes the rest of the nodes called Timing Slaves.

- FlexRay – is a new in-car communication protocol under development by FlexRay Consortium[4]. The communication protocol defines a topology which consists of point-to-point connection with active stars. It is used to target application such as X-by-wire and powertrain modules, which requires a deterministic and error-tolerant communication system. The protocol's main features include high data rates, time- and event trigger behavior, redundancy and fault-tolerance. The FlexRay protocol is also based on a time-triggered architecture where communication is organized in predefined time slots on the FlexRay bus. This ensures deterministic behavior with predefined latencies and avoids bus overloads.

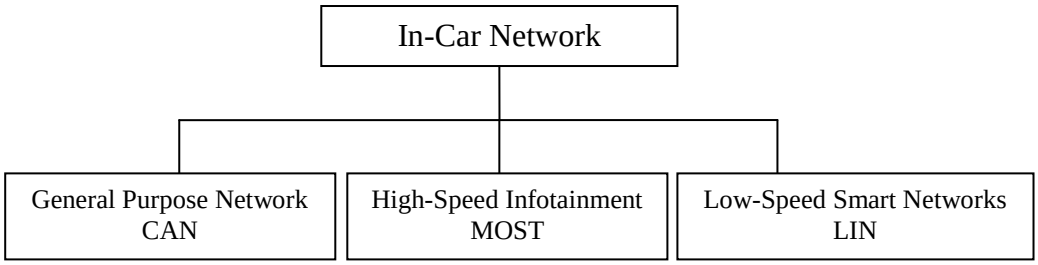


Chart 2.0 – Different types of in-car protocols.

2.7 Open Diagnostic Data Exchange

The Open Diagnostic Data eXchange format (ODX) is an XML-based Association for Standard of Automation and Measuring Systems (ASAM) standard for describing diagnostically relevant ECU data based on ISO 22901-1. A format to describe diagnostic information, functionality, and communication interfaces of in-vehicle ECUs. It covers the need of the entire vehicle life cycle from system engineering to the service shop. It is used to simplify the exchange diagnostic data between manufacturers, suppliers and service dealerships. It is also used to describe how to send and receive messages from individual bits or bytes together, how the messages are decomposed into

[3] Visit the homepage of http://www.mostcooperation.com/ for more information.

[4] Visit http://www.flexray.com/ for more information about FlexRay.

separate signals and values extracted, as well as how these values are transformed from a binary representation into a readable text [16][17].

The ODX data exchange format is primarily used to parameterize test systems. ODX data contain all information needed to diagnose ECUs and vehicles. This facilitates the creation of data-driven diagnostic applications. ODX also provides a modular system for diagnostic description. It supports many application cases, offers various methods for avoiding redundancy, and lets users take their specific requirements into account in describing the data [16].

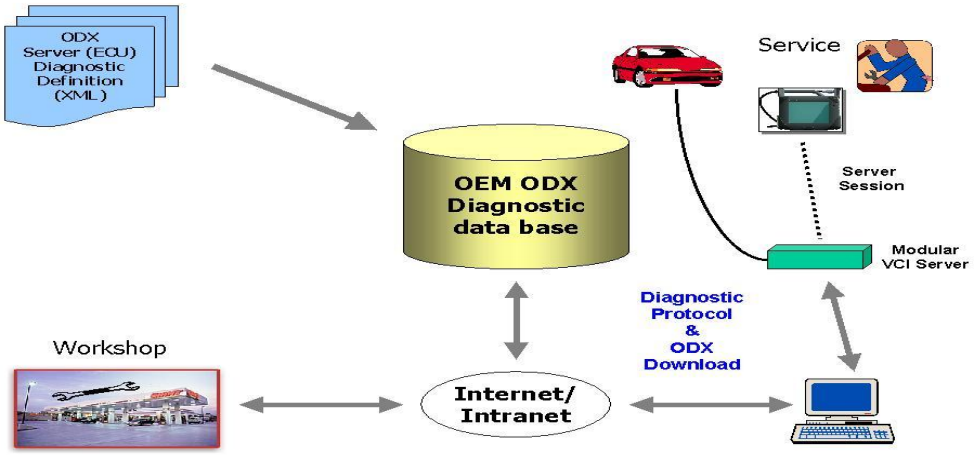


Figure 2.0 – ODX central source diagnostic data process.

The figure above shows the "central source" origin of diagnostic data, a verification and feedback mechanism with distribution to end-users. Engineering, manufacturing, and service specify which communication protocol and data shall be implemented in the ECU. This information will be documented in a structured format utilizing the XML standard. The XML file is used to setup the diagnostic engineering tools to verify proper communication with the ECU and to perform functional verification and compliance testing. Once all quality goals are met the XML file may be released to an OEM database. Diagnostic information is now available to manufacturing, service, OEM franchised dealers, and aftermarket service outlets via Intranet and Internet.

2.7.1 ODX Data Model

The ODX data model specification contains all diagnostic data to describe the data of a vehicle and physical ECU (e.g. diagnostic trouble codes, data parameters, identification data, input/output parameters, variant coding data, communication parameters, etc). ODX is described in UML diagrams and the data exchange format utilizes, as described in the previous section in XML [16].

The ODX modeled diagnostic data describe two diagnostic relevant parts of the vehicle:
a) Diagnostic data contained in the ECU, and;
b) Diagnostic relevant data required from the vehicle’s point of view.

The objective of this specification is to ensure that diagnostic data from any vehicle manufacturers is independent of the testing hardware and protocol software supplied by any test equipment manufacturer [16].

So the intention of the ODX data model is to contain all information that is required by a diagnostic tester to do a diagnostic communication with a specific ECU or set of ECUs. This means that all ECU specific parameters for communication are completely contained in the ODX and no ECU specific software in the tester software is necessary. This makes the tester completely data driven, and when testing new ECUs no software must be updated, just the appropriate ODX data must be loaded.

The ODX data modeled diagnostic data describe [16]:

- protocol specification for diagnostic communication of ECUs;
- communication parameters for different protocols and data link layers and for ECU software;
- ECU programming data (Flash);
- related vehicle interface description (connectors and pin-out);
- functional description of diagnostic capabilities of a network of ECUs;
- ECU configuration data (variant coding).

2.7.2 Example of an ODX file

```
<virtual>
  <ecu id="427">
    <request data="19020C">
      <response time="1219" data="59020C2357000C" />
    </request>
    <request data="14FFFFFF">
      <response time="1219" data="54FFFFFF" />
    </request>
  </ecu>
</virtual>
```

Example 2.0 – ODX file implemented in XML.

The scheme above shows a simple communication session between a vehicle and a diagnostic program. The session contains two request and the corresponding responses. Explanation of other similar schemes will be more described in chapter 3.

2.7.3 Other ODX Tools

There are several other ODX tools that can be used to edit a virtual platform. Although ODX is a standard format for vehicle diagnostic communication, it can be implemented in different ways. Therefore, there are no standard editors that can deal with all aspect

of configurations, because car manufacturers tend to make their own interpretation of a virtual platform.

However, there are several ODX viewers, such as CANdela studio or Intrepid ODX viewer, which allows a user to view files and search for specific items and to manually edit parameters inside those files. It must also be said that these viewers are in-house viewers and may not work for different virtual platforms based on ODX.

2.8 Virtual Car

Although discussed in Chapter 3, a virtual car is a converted read-out car log that contains information needed to perform certain functions when not working with a physical car. The intention with a virtual car is to be able to conduct certain tests that show different results before applying those tests to a physical car where hardware could get damaged. This way the designer/programmer/analyst can be certain that no failure could come to the ECU nodes if testing is applied to a virtual car first. The OEM defines a set of standards that a car manufacturer is supposed to follow during the test stage.

The development of distributed network-based systems often utilizes multiple suppliers for the prototyping of different modules and sub-systems. In order to best control the complexities incorporated from such a distributed developmental process, the Original Equipment Manufacturer usually requires a set of standard tests and procedures to be run on the prototypes prior to delivery. These tests usually require the prototype ECU to be connected to a simulated system where performance measurements can be made for consideration of the physical layer, communication layer, and application layer. The standard tests are run repeatedly until the Device under Test (DUT) passes all necessary tests. The requirements may include portions of the following sample test sequence:

- Voltage Characteristic Protection Tests
- Communication Waveform Characteristics
- Software Recovery from Error Conditions
- Diagnostics – which could include Recognition of faults

These examples may vary depending on the company that uses this facility [18][19].

3 Virtual Car

This chapter starts by looking into what a virtual car is as well as how it is created, together with other important issues such as the model data module and the service data module.

- A virtual car is an XML scheme of a car's physical electronic circuits and its accessories conforming to the ODX standard [17], converted from a read-out car log with a special converting program (see section 3.2).
- The model data module is an implementation specification of an ECU that specifies e.g. the name of the ECU and its different parameters (see section 3.3).
- The service data module specifies data link requirements of diagnostic services, which allow a diagnostic system to control diagnostic functions in an ECU, connected on a serial link embedded in a vehicle. It can be seen as the language spoken between a physical car and a diagnostic application and is based on ISO 14229-1:2006 [6] (see section 3.4).

Also in this chapter, the different modules used and needed are presented, starting with some background information. Finally, during the course of this chapter several user scenarios will be explained with examples from the editor.

3.1 Virtual Platform

The increasing electronic modules in vehicles are causing changes in the automotive electronics. One of the solutions made by the automotive companies was to connect several modules by lots of wires. Because the increased amount of modules in a vehicle, the embedded software will also increase in proportion. The increase in embedded software will lead to a change in software development methodology in order to control the growth of software that needs to meet reliability requirements. The growth in software content and the need for reliability from a finite supply of software engineers will lead to optimization problem. The vehicle manufacturers solved these problems by mounting the ECUs closely to whatever they are controlling. As for the reliability problems the vehicle manufacturers add redundancy for the most critical systems. But this method also bring drawbacks, because optimization like this can only be achieved by taking a more systematic approach to the electronic architecture, which in turn, requires measurable analysis, both of ECU networks and the internals of the ECU themselves. Another issue with the embedded software is the creation and testing of it, which further leads to productivity problems. Bench testing the software using hardware is becoming more and more difficult due to the difficulty of building hardware boards needed for performance issues.

A comprehensive solution has been developed by several automotive manufacturers, which allows solving the architectural analysis crisis while actually increasing software reliability – the *virtual platform* approach. A virtual platform is a fast simulation model

of an ECU or an entire network of ECUs (see section 2.3). The advantage of a virtual platform is allowing the full software load to be run and the accuracy enough to produce the true timing and the true network traffic that the real ECU subsystem will experience. This in turn will lead to a better visibility than the old bench-based approach, leading more quickly to software that is more reliable, and to an optimal architecture for ECUs [19].

3.2 Virtual Car

As described in chapter 2 section 2.6, a channel between the VCI and the car is established upon communication between the two after plugging in the VCI diagnostic connector to the vehicle. When communication is established, a log file is also created simultaneously which will log the communication between the diagnostic program, the VCI and the vehicle. The log file created is unique and a new log file is created for every vehicle that the diagnostic program communicates with, but if a log file already exists then the data written will be appended to the end of the existing log file [6]. When the diagnostic program reads out the car parameters with the help of the VCI, it may use different built-in scripts to perform several important functions, e.g. a script for reading out the VIN-number of the car, which is unique for every car. It might also load a script that checks the status of the ECUs in the car and whether they are active or not. Example of a very small part from a car log can look like:

```
--------------------------------------------------------------------------
1. Starting new logsession: 2008-08-26 10:38:58
--------------------------------------------------------------------------
2. Version info: Database 'Module Data', Version '080, 2008-08-05'
3. Database 'Service Data', Version '1.0, 2008-08-05'
4. [DiagnosticVehCom][Debug]   VehComm request: Ecu '726', Message '2203'
5. [J2534ChannelMana][Trace]   --->       (0) 00,00,07,26,22,03
6. [J2534ChannelMana][Trace]   <---       (0) 00,00,07,26,62,03,00,32,01,F4,
7. [DiagnosticVehCom][Debug]   VehComm response: '6203003201F4'
```

Text 3.0 – Communication between VCI and vehicle in a car log.

The communication between the vehicle and the diagnostic system as shown in Text 3.0 is based on question and answer model. The first line shows the time when the communication was established between the vehicle and the diagnostic system. The second and third lines shows the database modules which the log file uses to determine the size of different parameters, such as the size of a request item or the size of response data. These modules are also used to map to different service items needed to extract information from (see sections 3.4 and 3.5).

In the fourth line, the diagnostic system specifies the ECU that the question is to be directed to with a message that specifies a special task for that particular ECU. The question is indicated with "--->" and starts at the fifth line, shows the ECU specified and the parameter after the ECU which is supposed to perform a function to the ECU. The sixth line which is the answer, indicated with "<---", shows the answer with a

positive hex-value. The positive hex-value is taken from the question which is in this case 22. The hex-number 40, which is defined as a standard from ISO 14229-1, is added to the hex-value 22 if the answer is positive. If the answer is negative from the car the return hex-value will show 7F, which also is defined as a standard in ISO 14229-1. Following the hex-value 7F is information that states the fault or the problem. So the positive answer from the vehicle now corresponds to the hex-value 62. The last line shows the answer from the vehicle as a whole parameter with all the information needed to interpret the performance. This pattern may be repeated to the same ECU but with different question and parameters, and is also adapted to the other ECUs if they are to be interacted with.

As described, a virtual car can be created by different methods. Every step during the communication is logged in a log file which is created upon communication establishment by the diagnostic system. As also mentioned the diagnostic system can perform different diagnosis such as - reading out the stored DTCs in the memory of a specific ECU or all the ECUs. Another activity can be to read out how many programmed keys the car might have.

A special log-viewer program can convert this read-out car log to an XML document conforming to the ODX standard. This particular XML document contains all the necessary information about the car and its electronic units. With this generated XML file, a designer/programmer/analyst can make changes in different ECU parameters to use for fault tracing. To see the changes, a diagnostic system is commonly able to read in the XML file and display the values embedded inside of it. An example of a change in the XML file could be, changing a time-stamp for downloading software to a specific ECU and see whether the software was downloaded in a correct manner due to the time delay.

The XML file representing the virtual car contains from the read-out car log, only the ECU ids, in other words the variants of an ECU type, the ECUs requests, and the responses that belongs to each request [15]. It should be mentioned that the requests in the virtual car can be one of the following service types;

1. An Activation parameter which can be used in the diagnostic system for enabling e.g. the radio, a window button, the brakes in addition to many other things or reprogramming of an ECU. In the virtual car, the response of an activation parameter is presented to the user as answers with time-stamps and data. The response in this case can therefore be dynamic. These answers can then be used to test different activation scenarios.

2. A Parameter which can span from reading out the temperature of the engine to reading of values from the electrical system such as - current of a node, voltage, or the RPM.

3. A Default value which contains everything else that does not deal with activations, parameters. It may be DTC, how to interpret it or how create a DTC.

In the subchapter 2.7 in chapter 2, a small example was shown to the reader of how such a file can look like. Next, two additional examples, using XML-schema, will be presented and explained with some details.

Example 1- Showing a DTC request and response

```
<virtual>
  <ecu id="41">
    <request data="190406">
      <response time="1" data="5904062620FFFD22F" />
    </request>
  </ecu>
</virtual>
```

Example 1 – A scenario of a virtual car with ECU id, request and response data.

As shown in the example above, the XML document starts with a virtual tag which indicates that this is a virtual car both to the end-user and to the diagnostic system. The second tag shows the ECU variant of an ECU type, which could be a variant of an engine control module, a driver door module or maybe an automatic braking system. The request tags begins with the hex-value 19. The interpretation of the hex-value 19 lies in the service data. The value that follows the hex-value 19 gives an identifier that identifies a service used to set a data record or a group of records to specify the values on request from the end-user.

The response tag shows a time-stamp and the data and can also be thought of as the answer as seen in the car log. The data in the response tag shows the hex-value 59 which as indicated in Text 3.0, is a positive value, indicating a success in reading the DTC. The long string that follows the hex-value 59 holds different DTCs embedded in it. By looking up in the service data (see section 3.4), it is possible to see how the string is allocated to different items. To look up a DTC code one would have to use the model data (see section 3.3) to find a specific DTC based on the block item. This pattern of extracting information from the long response string, applies to all different service requests that the virtual car might have.

In Example 2 below, which starts similar to Example 1, shows the XML document starting with a virtual tag indicating to the end-user or diagnostic system this is a virtual car. The ECU id tag, shows a variant of an ECU type. Below the ECU id and its value is the request value, which as seen before is the question in the read-out car log. By looking into the values of the request data, the hex-value 22 is shown at the beginning of the request has also its interpretation in the service data (see section 3.4). The value that follows 22 is an identifier identifying different records for later mapping them to corresponding texts in the response data using the model data.

Following the request data is the response data which can be seen as the answer in the read-out car log. This particular response data is divided up in time, where the time shows a different sequence of parameters, for instance; the recorded speed of the car or different degree(s) read out during a specific sequence. This answer is said to be as a dynamic answer because time exceeds more than one unit. As explained in Example 1, the response data begins with the hex-value 62 which is the positive answer from the car.

The hex-value following 62 is the service request (2610) as also seen in the request data. The data that proceeds the service request data is divided up in several blocks, but the difference between the DTC response data and the parameter response data is, the parameters interpretation lies in the model data instead of the service data. Altough the service data specifies the diagnostic communication, it is still used to show the positive answer of a request and other attributes. The same pattern also follows here for mapping to the model data and finding out the embedded values, but the model data also gives an offset as the starting position to find data needed.

Example 2 – Showing a Parameter request and response

```
<virtual>
 <ecu id="60">
  <request data="222610">
   <response time="8588" data="622610DFF7DFF15993F993" />
   <response time="12194" data="622610FFFDFF7DFF15774" />
   <response time="12881" data="622610FFFDFF7DE581488" />
   <response time="13475" data="622610FFFDFF7DE0B44FEB" />
  </request>
 </ecu>
</virtual>
```

Example 2 – An example that shows a Parameter inside a virtual car.

3.3 Model Data

The model data, which is an XML document, can be seen as a database for storing information about the ECUs and its parameters, in contrast to the service data that shows how communication is set up. The model data gives a lot information about the ECUs in thevirtual car, for instance; information about all the DTCs stored, their names and other information regarding DTCs. Information about the ECU types, their names and one of the more important things, how many ECU variants each ECU type might include. It also shows information for determining what kind of a service each ECU variant is carrying. A service that might be reading out DTCs or parameter values. Because communication can be carried out in different protocols, it also displays the protocol id for each ECU variant (see chapter 2 section 2.6.1). Because the file is structured in XML way, what follows the ECU type tag will be its' child. The child of

an ECU type is an ECU variant. As explained before, an example of a variant can be an engine variant or a braking system variant. The ECU variant is important because it is here the underlying information will be displayed that will allow the end-user to find information about the different DTCs, parameters and other service requests. The information could be the unique id of an ECU variant, its' name and the protocol id which it uses when communicating with the car. If an ECU is not active during communication with the car, the read-out car log will only log the ECU variants name and id. If active, the ECU variant will in the XML file have children appended to it, depending on what information is stored in the memory of the ECU. The children are used to find out if the request data is a DTC, a record, a parameter or a routine and how to interpret the data.

3.4 Service Data

As mentioned in the introduction of this chapter, the service data is a specification of the communication that takes place between the car and the diagnostic system. The service data shows different meanings for the service request, e.g. how to interpret the request and the response. It is in service data where the implementation information is found about a certain ECU and its' data. The service data also shows if a parameter can be a parameter, an activation, or a default value. This is then mapped into information that tells whether these values are a DTC, a parameter, a record, or a routine.

The hierarchy of the service data is built up by first showing the protocol used for communication. Therefore, it is important to find out what protocol is used in the virtual car before extracting information from the service data. The protocol used in the virtual car is displayed as in the model data. The hierarchy is then divided into two parts, a request part, that gives information about how to interpret requests and a response part that shows how the response data is supposed to be interpreted.

3.4.1 Request and Response in Service Data

The request and response tags do not provide any information except for telling where each section starts. The request will always have two children attached to it. The first child attached to the request is a service type, which shows what value the request data begins with; for instance the hex-value 19 or 22 (see Examples 1 and 2). The second child might vary between a block item and a mode item. A block item is used in every context except for when it comes to DTCs, where mode item is used. The block item shows the unique id attached to the block item, its name, and most importantly the length of it. Other information it gives is the block-type, which again can be a DTC, a record, a parameter, or a routine. A second child can be the mode item. It should be stated that only one of the block item or mode item is followed the service item. The mode item functions primarily like the block item but instead of a block-type the mode item possesses an order which defines how the data is to be structured.

Example 5 – Showing hierarchy with block item and with mode item

Scenario 1 – block item

```
<protocol>
  <service>
    <request>
      <service item />
      <block item />
    </request>
    <response>
    </response>
  </service>
</protocol>
```

Scenario 2 – mode item

```
<protocol>
  <service>
    <request>
      <service item />
      <mode item />
    </request>
    <response>
    </response>
  </service>
</protocol>
```

Example 5 – Hierarchies with block item and mode item.

The response tag that follows the request tag has similar children but, it also have either a fixed list or a dynamic list as a child. If the request data is a DTC query then the fixed list will be applied to the response data, because the response will only have one answer back. The dynamic list applies to parameter responses and such, if the response data spans more than one answer.

In the Example 6, it is shown how the structure can be seen when working with a dynamic list or a fixed list. The dynamic list in the response follows the block item and the fixed list is followed by the mode item. Because the fixed list is applied to a fixed response data, it also has children. A block item that shows e.g. a DTC or any other thing that has a fixed block size. The second child that follows a block item is a skip item. A skip item is used for marking up unused characters that follows after a block item or a mode item but is necessary for definition in the response data string. The block item under the fixed list does have the same properties as other block items that are not a child to the fixed list tag. It shows a unique id, a name and most importantly the length of the block.

The dynamic list gives some information, such as the unique id and order id that defines in what order the elements in the dynamic list is in. The difference between the fixed list and the dynamic list is that model data must be used for mapping blocks when working with dynamic lists. The model data gives information about how to cut up the blocks in the dynamic list, with the help of offsets. The offset gives a starting position and a correspondent length to that particular offset.

Example 6 – Showing dynamic and fixed lists

Scenario 1 – dynamic list

```
<protocol>
  <service>
    <request>
      <service item />
      <block item />
    </request>
    <response>
      <service item />
      <block item />
      <dynamic list />
    </response>
  </service>
</protocol>
```

Scenario 2 – fixed list

```
<protocol>
  <service>
    <request>
      <service item />
      <mode item />
    </request>
    <response>
      <service item />
      <mode item />
      <fixed list />
    </response>
  </service>
</protocol>
```

Example 6 – Showing hierarchies with a dynamic and a fixed list.

4 Software Design and User Interface

An important part of this project was to develop an editor so that the end-user can easily edit a virtual car. In this chapter I will present the software design and implementation of the virtual car editor. We will also explore how different classes are coupled and tied together as well as how different problems were tackled and what approaches were used for solving these problems. Throughout the text, graphical user-interfaces implemented and the function of each form will be described.

4.1 Program Overview

Working with the virtual car has so far been very inconvenient, because the programmer/analyst/technician or any one working with the virtual car have to edit the parameters and data manually. With this virtual car editor it is now possible for the end-user to edit any aspect of the virtual car by the forms developed without having to browse around in different files needed for editing.

This implementation of the virtual car editor is divided up in four parts.

1. The actual text editor is the first part and enable end-users to read in a virtual car file as well as importing the correspondent model and service data files. This main form has a menu-bar that allows the end-user to perform various actions, such as opening a file, saving a file among other things. In this main form there are also three buttons for displaying other forms that will aid in editing of the virtual car.

2. The ECU editor form which allows editing of ECU variants in the virtual car is the second part, and is accessible through a button in the main form. The ECU editor form will display the ECU ids in the virtual car in a combo-box and listing the ECU ids in the model data in another combo-box for easier replacements.

3. The DTC form for editing DTCs is the third part. Accessing the DTC form is also done through a button from the main form. In the DTC form, several buttons can be used for helping the end-user with his/her change of data. In addition, the DTC form also contains several combo-boxes and list-boxes allowing the end-user to select the proper data needed. Adding, replacing and deleting DTCs are the options given when editing the DTCs in the virtual car.

4. The parameter form that allows editing of parameters in the virtual car is the fourth part. The parameter form will help the end-user to do all the calculations necessary when editing parameter. Also, instead of having the end-user calculating on offsets with dynamic lists that may occur (see chapter 3 subchapter 3.4), the form will extract all the necessary data and do all the calculation, showing the end-result to the end-user in an understandable manner. In this form the end-user has the option to create new requests and responses based on information from the model and service data.

Next in this chapter, examples of how these forms look like will be presented along with explanation to each form.

4.2 User Interface

This subchapter will explain the different forms implemented as well as the design of the forms and what they are intended to do. One of the difficult parts of this thesis was to make a good design for the forms to be used by the end-users. As GUI needs to be comprehensible, I had to think carefully of how to design the different forms. I conducted a few interviews among the engineers of the company I was stationed at and came up with the solutions presented.

4.2.1 Main Form

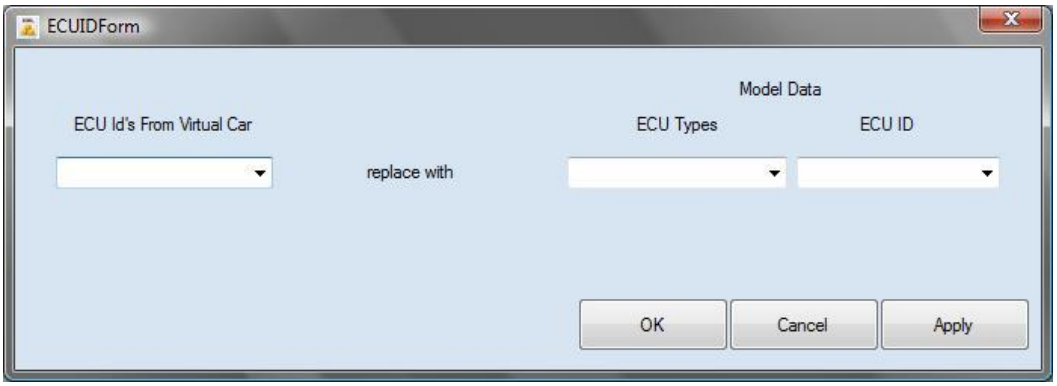
The main form is an ordinary read-only text editor that supports an XML file based on the ODX standard. It also features different buttons that displays other forms the end-user needs to edit the virtual car. The ECU ID form for editing ECU ids embedded inside the virtual car. The DTC form for editing DTCs in responses and a Parameter form for configuring parameter data also inside responses. It also allows the end-user to import the model and service data specified for this particular virtual car.

Picture 4.0 – Shows the main form used to read in the virtual car.

4.2.2 ECU ID Form

The ECU ID form is used to replace the existing ECU variants or ids embedded in the virtual car with ECU ids that are available in the model data. As Picture 4.1 shows, there are 3 different combo-boxes. The left combo-box will contain the ECU ids from the virtual car. The center combo-box contains all the ECU types that exist in the model data file, such as Engine Control Module, Central Electronic Module, among others. The right combo-box which is activated after selecting an ECU type is the one that will display the ECU variants under each ECU types. So if the ECU type's combo-box

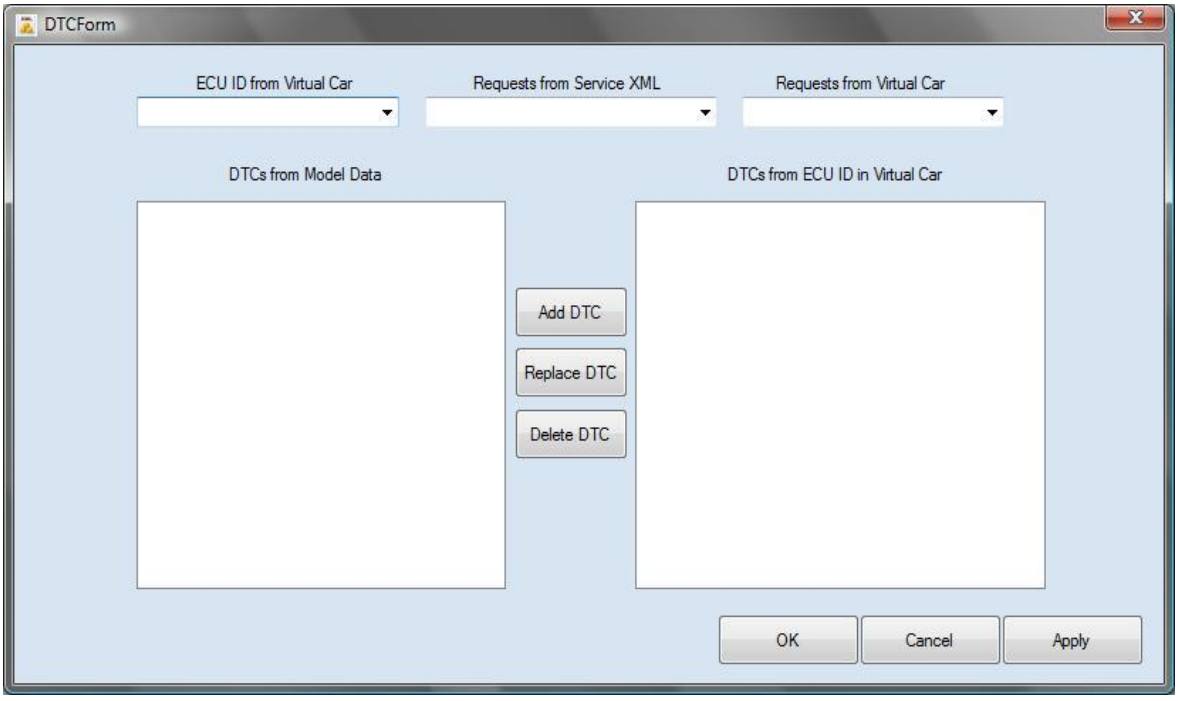
displays for instance Engine Control Module, the ECU id combo-box will contain all ECU ids under that specific ECU type. It could be a variant such as Bosch or Denso.



Picture 4.1 – The replace form used for editing ECU ids in the virtual car.

4.2.3 Diagnostic Trouble Codes Form

The DTC form enables the end-user to edit DTCs that are stored in a response of a correspondent request in the virtual car. The request is a query to an ECU about one or several DTC(s) and the response contains the DTCs that the end-user can edit.



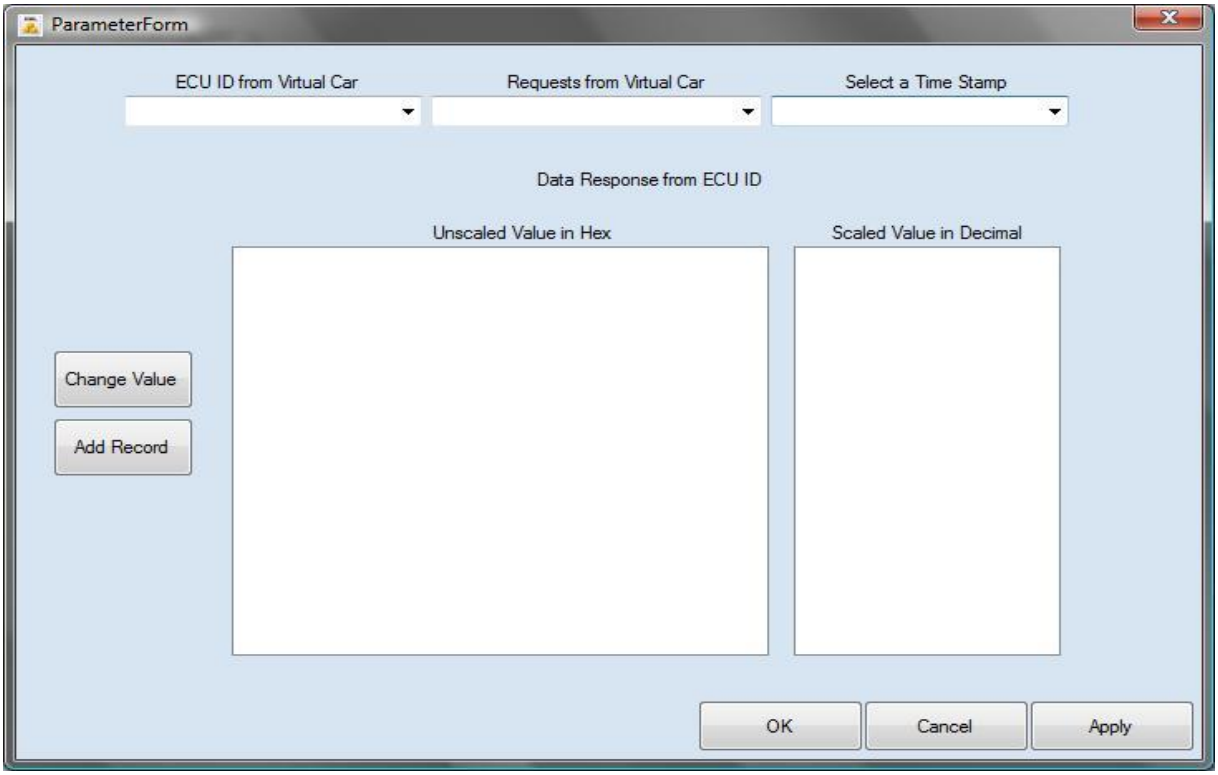
Picture 4.2 –DTC form used by the end-user to edit DTCs.

As shown in Picture 4.2 there are several combo-boxes. The left combo-box will contain the ECU ids from the virtual car, and once the user selects an ECU id, the list-box under “DTCs from Model Data” will display DTCs from the model data that exist under the selected ECU id. This is done by having an event listener listening for changes in the combo-box, in other words a callback method. The center combo-box

will contain services that deal with a DTC. The right combo-box will now contain all the requests under the respective service, but these requests will come from the virtual car. After selecting a request from the virtual car, the list-box under "DTCs from ECU ID in Virtual Car" will display the DTCs embedded in the response data under the selected request.

4.2.4 Parameter Form

The Parameter form, will allow the end-user to edit parameters in the virtual car as well as create a new request with its correspondent response. As described in chapter 3 section 3.3, the parameters are made up by a time-span and can be dynamic. Therefore, the parameters are shown by choosing a time-stamp.

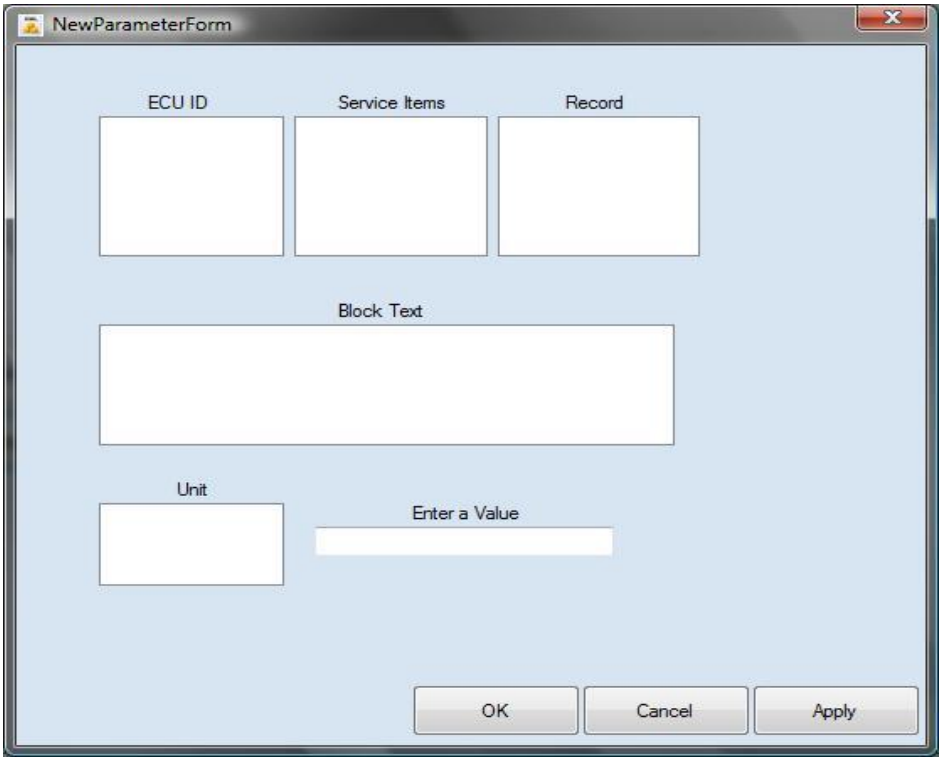


Picture 4.3 – Parameter form used to edit and create parameters.

As described, when dealing with parameter data an offset is set to extract certain information. It could for instance be rotation per minute, voltage or current of a node. Because this piece of information is given in hex-value (the first list-box will display this value), it is made into decimal format or into a bit if the offset data might be an option value. After making the hex-value into a decimal number, a formula is also given for scaling the decimal to the right value, which will be displayed in the second list-box. The Add Button allows the end-user to create a new parameter from scratch, both with request and a correspondent response. The data needed to create a new parameter is extracted from the model data. The Change Button allows the end-user to change the hex-value in the un-scaled list-box or to choose a new bit.

4.2.5 New Parameter Form

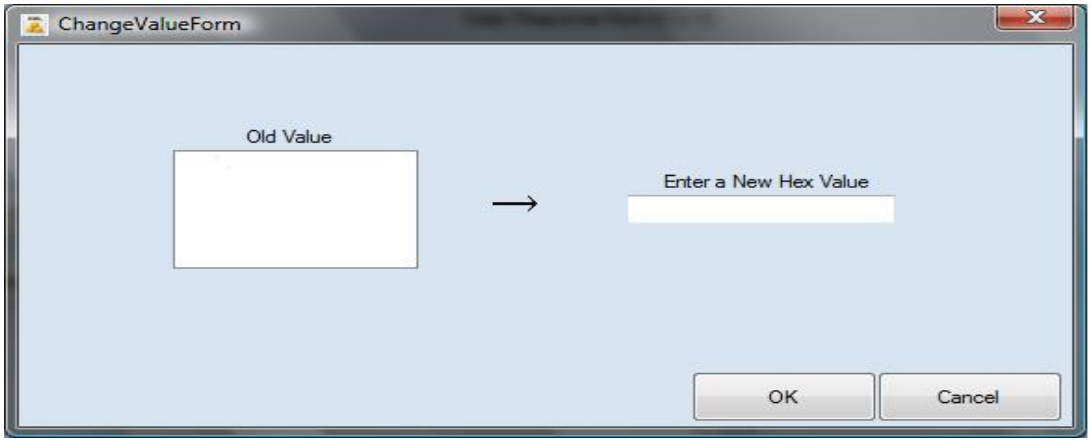
In the New Parameter form the end-user is able to add a whole new request and response under a selected ECU variant/id. The user gets the option to choose from the different parameters extracted from the model data, displayed in the list-boxes shown in Picture 4.4. Although the simplicity, the end-user must type in values to be created for the response data tag.



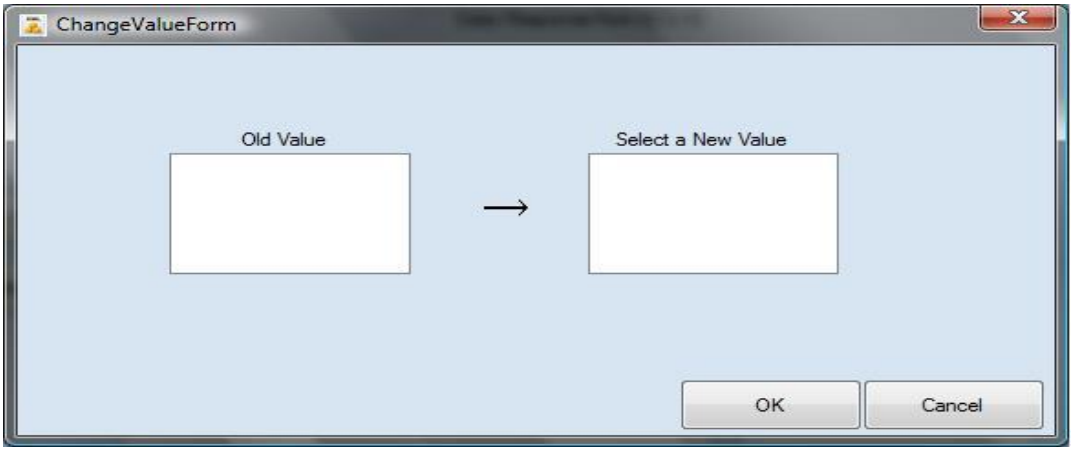
Picture 4.4 – A form for creating requests and responses.

4.2.6 Change Value Form

The Change Offset Form has two functions. It allows the end-user to either change a hex-value or a bit-value from a binary string. This bit-value represents either on/off, true/false, yes/no, so called option values. The block item after the service item determines if the offset is a hex-value or a bit-value. The bit-value is extracted from an offset where the offset shows a part of a string, usually one character and this character is then transformed into a binary format string. The bit that is extracted is determined by the offset.



Picture 4.5 – Change Offset Form, allows the end-user to edit a hex-value.



Picture 4.6 – Change Offset Form, allows the end-user to change a bit.

4.3 Software Design

In this subchapter I will explain what methods and approaches I used to solve different problems when developing this application.

4.3.1 ModelCommon and ServiceCommon

A lot of information that is looked up in model data and service data are the same when extracting data from these files. Therefore, a common class that deals with model data has been created for classes that have many things in common. An example of this is, in model data many attributes are named the same for many types. Hence, I have created a class called ModelCommon to store these common attributes and let other classes inherit from this class. For instance, a block item and its data is the same everywhere in the model data so only one class has been created for this purpose. As it appears in Figure 4.0, what is inherited from the ModelCommon class is the attributes name and ident (-ification).

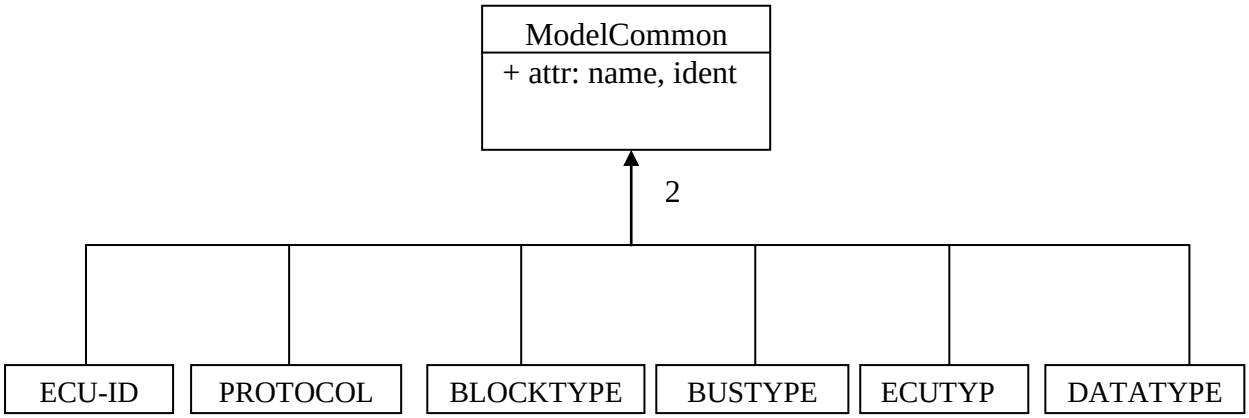


Figure 4.0 – Showing how different classes with same data (name and ident) inherits attributes from the ModelCommon class.

Figure 4.0 adopts the Model-View-Controller pattern, where the ModelCommon class is the Model part. The model part can be thought of as a domain-specific representation of the information on which the application operates. It adds meaning to the raw data. The classes inheriting the attributes from the ModelCommon class can be seen as the View part. The view part is responsible for presenting data to the user through a combination of graphics and text. The controller part processes the data when the user reads in the model data file. The controller is the means by which the user interacts with the application. The controller accepts input from the user and instructs the model and/or the view part to perform actions based on that input.

The same figure as Figure 4.0 could also be applied to service data where a lot of parameters extracted have attributes that are the same. Although, the names of the classes that inherit from the ServiceCommon class differs. The attributes or the variables that the classes inherits from ServiceCommon are also name and ident (-ification) attributes.

4.3.2 Forms Structure

When first starting the editor, a blank text editor will appear in front of the end-user at the screen. The end-user will then have the option to read in the virtual car as well as the correspondent model and service data. When reading in the virtual car, it is stored in an arraylist data structure to preserve the hierarchy. The hierarchy of the virtual car is important to keep because, when creating a new parameter it is important to know where to create it and how to look for the same record, if creating a new response under an existing parameter.

As for the model and service data, when the two files are read in, their hierarchy are stored in a respective hash table. The hash table are used for fast look-ups which makes it easier to find an element. In this case, when dealing with the model and service data the hierarchy does not need to be in order because the files are not modified but only read from. The model and service data has each a class for doing these operations.

Storing the model and service data in classes with help of hash tables is yielded by the forms when required to look up information in the files for applying some changes.

The ECU ID form as described before is used to edit the different ECU ids inside the virtual car. The replace form when opened reads in the ECU ids with the help of the arraylist that stores the virtual car. It also uses the hash table in the class that stores the model data as it needs to display the different ECU types and ids embedded. The other remaining forms uses the same technique when uses the arraylist and the hash tables.

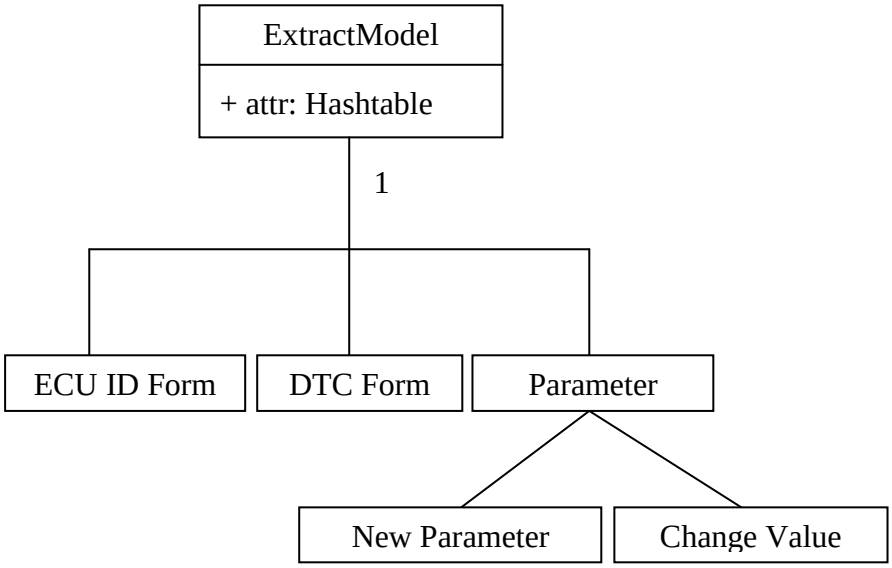


Figure 4.1 – Showing the association between the different forms and the extract model class.

Figure 4.1 shows the association between the forms that enables the end-user to edit the virtual car, and the extract model class that stores the model data file. The service data file is used in similar manners. A class named ExtractService is used to store the service data file. The ECU ID, DTC and New Parameter forms uses the extract service class in the same way for finding relevant information.

4.3.3 Save Classes

For easier information finding when dealing with data from a virtual car, I have created several classes that stores different information about pieces of data from the virtual car. For instance, the interpretation of a DTC data embedded in the virtual lies in the model data, data that can be the length of the DTC, its name, its unique id among other things. Because DTC data from the virtual car is displayed in a list-box, extracting information regarded to this piece of DTC is now simple. By selecting a DTC in a list-box, information about this particular DTC will be easily extracted.

5 Conclusions

5.1 Results

The purpose of this thesis work was to program a virtual car configurator. The configurator provides options to edit several different parameters embedded in a virtual car with help of databases used for storing information about the virtual car. Throughout the way of this work, new ideas and modifications were made to enhance the performance and the design of the configurator.

The final result of this configurator is now used at Sörman Information AB. After conducting a few surveys' about the usage of the configurator, technicians find it much easier working with the editor, because of not having to manually change parameters in a virtual car.

Diagnostic requests and responses can now also successfully be created and read in by the configurator for further analysis.

Support for error handling have been carefully thought of and allows the end-user to work without having to handle any errors.

This virtual car configurator can now be used as a base in future improvement or add-ins to make it more suitable for other types of virtual cars not conformed to the ODX standard.

5.2 Future Work

Reduction of cyclomatic complexity in some part of the program might be made to further optimize the code and the performance of the configurator. Other improvements might be to allow this virtual car editor to handle any type of virtual car that is not extracted from UpTime System[5] as the diagnostic program, and with other model and service data.

Possible changes in reading in the XML files can be reviewed using serialized classes instead of regular XML streams for faster memory access. This gives the option once again to enhance the code and to decrease the cyclomatic complexity.

Other improvements can be allowing the end-user to create a dynamic response when creating a new parameter, by generating a time-stamp that spans over a certain time and also letting the user create option value such as true/false, yes/no etc. that enables binary editing of parameters.

[5] Please visit http://www.sorman.com/products/uptime/index.asp?l=en for more information on UpTime Systems.

Terms and Abbreviations

ASAM	Association for Standardization of Automation and Measuring Systems. An association that provides standards for data model, interfaces and syntax specification for a variety of applications, such as testing, evaluation and simulation.
CAN	Controller Area Network. Serial communication with emphasis on reliable transmission. Used in vehicles.
DTC	Diagnostic Trouble Code. An error code logged by a control unit in the vehicle. Indicates what is wrong.
ECU	Electronic Control Unit. A node in the network of the vehicle.
GUI	A graphical user interface to a computer.
ISO	International Organization for Standardization. An organization for developing and publishing international standards.
ISO 11898	Standard that specifies a serial communication technology called Controller Area Network.
ISO 14229-1	A general set of diagnostic services.
ISO 22900-1	Standard that specifies how a vehicle communication tool works.
ISO 22901-1	Standard that describes a new technology called Open Diagnostic Data Exchange.
LIN	Local Interconnect Network. A vehicle bus standard used within current automotive network architectures.
MIL	Malfunction Indicator Lamp. An indicator of the internal status of a car engine.
MOST	Media Oriented Systems Transport. An electronic bus type architecture for on-board audio-visual devices. Used primarily in automobiles.
OBD	On Board Diagnostics. An emission related set of diagnostic services.
ODX	Open Diagnostic Data Exchange. An XML-based ASAM standard for describing diagnostically relevant ECU data.
OEM	Original Equipment Manufacturer. Typically a company that uses a component made by a second company in its own product.
SDK	Software Development Kit. A software packet to be used when new software shall be developed.

UML	Unified Modeling Language. An object modeling and specification language used in software engineering. Includes a set of graphical notation techniques to create abstract models of specific systems.
VCI	Vehicle Communication Interface. A diagnostic tool that communicates with the vehicle control units.
VIN	Vehicle Identification Number. A unique serial number used by the automotive industry to identify individual motor vehicles.
XML	Extensible Markup Language. A general-purpose specification for creating custom markup languages.

References

[1] Sörman Information AB, http://www.sorman.com, March 2009.

[2] Microsoft Corporation, *the C# Language*, http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx, March 2009.

[3] Microsoft Corporation, *.NET Framework Conceptual Overview*, http://msdn.microsoft.com/en-us/library/zw4w595w.aspx, March 2009.

[4] Microsoft Corporation, *Visual Studio 2008*, http://www.microsoft.com/visualstudio/en-us/default.mspx, March 2009.

[5] B&B Electronics, *OBD-II Background Information*, http://www.obdii.com/background.html, January 1 1996.

[6] International Organization for Standardization, *Road vehicles – Diagnostic Systems – Part 1: Diagnostic services*, ISO 14229-1, May 31 2001.

[7] SAE International. *On-Board Diagnostics for Light and Medium Duty Vehicles Standards Manual*. Pennsylvania, ISBN 0-7680-1145-0, 2003.

[8] U.S. Environmental Protection Agency, *OBD – Basic Information*, http://www.epa.gov/otaq/regs/im/obd/basic.htm, August 14 2007.

[9] Fresh Patents, *Vehicle Diagnostic System*, http://www.freshpatents.com/Vehicle-diagnostic-system-dt20080724ptan20080177438.php, USPTO Application #: 20080177438, June 24 2005.

[10] ECU Testing, *Engine Control Unit*, http://www.ecutesting.com/ecu.html, December 15 2008.

[11] Kassakian, J.G; Wolf, H.-C.; Miller, J.M.; Hurton, C.J.; *Automotive electrical systems circa 2005 - IEEE Spectrum*, http://www.spectrum.ieee.org/print/1420, 10.1109/6.511737, 2005.

[12] Free Patents Online, *Electronic control unit for automotive vehicles*, http://www.freepatentsonline.com/6243629.html, Patent 6243629, May 06 2001.

[13] Free Patents Online, *Vehicle Communication Control Apparatus and Method*, http://www.patentstorm.us/patents/6321148/description.html, Patent 6321148, November 20 2001.

[14] Ciulla, Vincent T.; http://autorepair.about.com/cs/generalinfo/l/bldef_154a.htm, *Diagnostic Trouble Code*, 2002.

[15] Miller, Jason; *Generic Global Diagnostic Specification*. Part 1. Doc. No. 31810161. May 2003.

[16] Reul, David A.; Raichle, Kurt R.; *Multi-vehicle Communication Interface,* http://www.patentstorm.us/patents/6526340/description.html, Patent 6526340, February 25 2003.

[17] Dr. Augustin, *Data Model Specification ASAM MCD-2D (ODX),* Version 2.1.0., September 15 2006.

[18] Mullery, Kevin; Jackman Brendan, *Experiences with the ODX Diagnostic Database Standard.* Doc. No. 2006-01-1568. April 2006.

[19] Mcellan, Paul; *Virtual platforms speed automotive design,* http://www.eetimes.com/news/design/columns/eda/showArticle.jhtml?articleID=17408738. October 17 2003.

[20] CAN in Automation, *Controller Area Network (CAN),* http://www.can-cia.org/index.php?id=46, 2001.

[21] Kvaser AB, *CAN (Controller Area Network),* http://www.kvaser.com/can/, March 2009.

Appendix A – Product Specification

This appendix will entitle the product specification requested from Sörman Information AB. It will describe what Sörman wants with this project and to what use.

Analysis

The task is to analyze whether an electronic unit data from a sample file in the standardized ODX format could be used to create a file in XML format which can be used to simulate electronic unit data. Timing aspects should also be included, i.e. different responses to be simulated during a time interval.

Specification

If the analysis is possible, a specification that describes the technical functionality of a configurator is to be implemented. Furthermore, user specified functions shall be regarded to create a simulation file using the configurator. This means that requirement gatherings should be carried out with prospective users.

Implementation

Development of a configurator that allows configuring simulation files various fault codes, parameters value, etc, from the content of an ODX file.

Test and Verification

The final functions of the configurator are to be tested in a real environment.

Prerequisites

Sörman undertakes the responsibility of providing an example file in ODX format from a subcontractor in the automobile industry. Sörman also undertakes the implementation of a function in UpTime that allows the created simulation file to be read in.